(Optional) Advanced Migrations

Database Structure as Code

Database Changes: Then vs Now

Current Approach:

```
-- Manually run SQL commands
CREATE TABLE students (id INT PRIMARY KEY...);
ALTER TABLE students ADD COLUMN gpa DECIMAL(3,2);
DROP TABLE students;  -- Oops! Lost all data!

-- Share changes via email or documentation
-- "Hey everyone, please run these SQL commands..."
-- Team members have different database structures
```

Laravel Migrations:

```
# Version-controlled database changes
php artisan make:migration add_gpa_to_students_table
php artisan migrate          # Apply changes
php artisan migrate:rollback # Undo changes safely
```

Database evolution made safe and trackable!

Database Migrations in Laravel

What Are Migrations?

- **Version control for your database**
- Write database changes in **PHP files**, not manual SQL
- Allow team members to share & sync database structure

Why Use Migrations?

- Keep **schema consistent** across environments (local, staging, production)
- Easy to **track changes** in Git
- Can **rollback** if something breaks
- Automate DB setup → `php artisan migrate`

Your database schema becomes code!

Creating Migrations

Create Migration:

```
# Create new table
php artisan make:migration create_courses_table

# Modify existing table
php artisan make:migration add_gpa_to_students_table --table=students

# With model creation
php artisan make:model Course -m
```

Generated Migration File:

```php
<?php
// database/migrations/2024_01_15_100000_create_courses_table.php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;
```

Migration Commands

Running Migrations:

```
php artisan migrate                    # Apply all pending migrations
php artisan migrate:rollback           # Undo last batch
php artisan migrate:rollback --step=2  # Undo last 2 batches
php artisan migrate:reset              # Rollback all migrations
php artisan migrate:refresh            # Reset and re-run all
php artisan migrate:fresh              # Drop all tables and migrate
```

Check Migration Status:

```
php artisan migrate:status
```

Output:

```
+------+-----------------------------------------+-------+
| Ran? | Migration                               | Batch |
```

Common Migration Operations

Column Types:

```php
Schema::create('students', function (Blueprint $table) {
    $table->id();                              // BIGINT UNSIGNED AUTO_INCREMENT
    $table->string('name');                    // VARCHAR(255)
    $table->string('email', 100)->unique(); // VARCHAR(100) UNIQUE
    $table->text('bio')->nullable();           // TEXT, can be NULL
    $table->integer('year');                   // INT
    $table->decimal('gpa', 3, 2);             // DECIMAL(3,2) for 0.00-9.99
    $table->boolean('is_active')->default(true); // BOOLEAN DEFAULT TRUE
    $table->date('enrollment_date');           // DATE
    $table->json('metadata');                  // JSON column
    $table->timestamps();                      // created_at, updated_at
});
```

Modifying Columns:

```php
Schema::table('students', function (Blueprint $table) {
    $table->string('email', 150)->change();    // Change length
```

Database Relationships

Your Manual Approach:

```php
// Get student's courses manually
function getStudentCourses($studentId) {
    $mysqli = new mysqli("localhost", "root", "password", "db");

    $sql = "SELECT c.* FROM courses c
            JOIN student_courses sc ON c.id = sc.course_id
            WHERE sc.student_id = ?";

    $stmt = $mysqli->prepare($sql);
    $stmt->bind_param("i", $studentId);
    $stmt->execute();

    return $stmt->get_result()->fetch_all(MYSQLI_ASSOC);
}
```

Laravel Relationships:

One-to-Many Relationship

Example: Student has many Enrollments

Migration for Enrollments:

```
Schema::create('enrollments', function (Blueprint $table) {
    $table->id();
    $table->foreignId('student_id')->constrained()->onDelete('cascade');
    $table->foreignId('course_id')->constrained()->onDelete('cascade');
    $table->decimal('grade', 3, 2)->nullable();
    $table->timestamps();
});
```

Models:

```
// Student Model
class Student extends Model
{
    public function enrollments()
```

Using One-to-Many:

```php
// Get all enrollments for a student
$student = Student::find(1);
$enrollments = $student->enrollments;

// Get student from enrollment
$enrollment = Enrollment::find(1);
$student = $enrollment->student;

// Create new enrollment
$student = Student::find(1);
$student->enrollments()->create([
    'course_id' => 2,
    'grade' => 3.5
]);
```

No manual foreign key management!

Many-to-Many Relationship

Example: Students belong to many Courses

Pivot Table Migration:

```
Schema::create('course_student', function (Blueprint $table) {
    $table->id();
    $table->foreignId('course_id')->constrained()->onDelete('cascade');
    $table->foreignId('student_id')->constrained()->onDelete('cascade');
    $table->timestamp('enrolled_at')->useCurrent();
    $table->decimal('grade', 3, 2)->nullable();
});
```

Models:

```
// Student Model
class Student extends Model
{
    public function courses()
```

Using Many-to-Many:

```php
// Get student's courses
$student = Student::find(1);
$courses = $student->courses;

// Get the course's students
$course = Course::find(1);
$students = $course->students;

// Attach course to student
$student->courses()->attach(1, ['grade' => 3.5]);

// Detach course from student
$student->courses()->detach(1);

// Sync courses (replace all existing)
$student->courses()->sync([1, 2, 3]);

// Access pivot data
foreach ($student->courses as $course) {
    echo $course->pivot->grade;
    echo $course->pivot->enrolled_at;
}
```

Advanced Relationship Features

Eager Loading (Solve N+1 Problem):

```php
// Problem: N+1 queries (inefficient)
$students = Student::all();
foreach ($students as $student) {
    echo $student->courses->count();  // Query for each student!
}

// Solution: Eager loading (2 queries total)
$students = Student::with('courses')->get();
foreach ($students as $student) {
    echo $student->courses->count();  // No additional queries!
}

// Load multiple relationships
$students = Student::with(['courses', 'enrollments'])->get();

// Conditional eager loading
$students = Student::with(['courses' => function ($query) {
```

Polymorphic Relationships

Example: Comments belong to Students OR Courses

Migration:

```
Schema::create('comments', function (Blueprint $table) {
    $table->id();
    $table->text('content');
    $table->morphs('commentable');  // Creates commentable_id & commentable_type
    $table->timestamps();
});
```

Models:

```
// Comment Model
class Comment extends Model
{
    public function commentable()
    {
        return $this->morphTo();
    }
}
```

Using Polymorphic Relationships:

```php
// Create comment on student
$student = Student::find(1);
$student->comments()->create(['content' => 'Great student!']);

// Create comment on course
$course = Course::find(1);
$course->comments()->create(['content' => 'Excellent course!']);

// Get all comments regardless of type
$comments = Comment::all();
foreach ($comments as $comment) {
    echo $comment->commentable_type;  // "App\Models\Student" or "App\Models\Course"
    echo $comment->commentable->name; // Student name or Course name
}
```

Real-World Example: Complete Student System

Models with Relationships:

```
// Student Model
class Student extends Model
{
    protected $fillable = ['name', 'email', 'major', 'year'];

    public function enrollments()
    {
        return $this->hasMany(Enrollment::class);
    }

    public function courses()
    {
        return $this->belongsToMany(Course::class, 'enrollments')
                    ->withPivot('grade')
                    ->withTimestamps();
    }

    public function getGpaAttribute()
    {
        return $this->enrollments()->avg('grade');
    }
}

// Course Model
class Course extends Model
{
    protected $fillable = ['code', 'name', 'description', 'credits'];

    public function students()
    {
        return $this->belongsToMany(Student::class, 'enrollments')
                    ->withPivot('grade')
                    ->withTimestamps();
    }

    public function enrollments()
    {
```

```php
// Enrollment Model (Pivot with additional data)
class Enrollment extends Model
{
    protected $fillable = ['student_id', 'course_id', 'grade'];

    public function student()
    {
        return $this->belongsTo(Student::class);
    }

    public function course()
    {
        return $this->belongsTo(Course::class);
    }
}
```

Complex Queries with Relationships

Using Relationships in Queries:

```php
// Students taking Computer Science courses
$students = Student::whereHas('courses', function ($query) {
    $query->where('code', 'like', 'CS%');
})->get();

// Courses with more than 10 students
$courses = Course::withCount('students')
                 ->having('students_count', '>', 10)
                 ->get();

// Students with a GPA above 3.5
$students = Student::whereHas('enrollments', function ($query) {
    $query->havingRaw('AVG(grade) > 3.5');
})->get();

// Load students with their courses and grades
$students = Student::with(['courses' => function ($query) {
```

Migration Best Practices

1. Naming Conventions:

```
# Table creation
create_students_table

# Adding columns
add_gpa_to_students_table

# Removing columns
remove_deprecated_from_students_table

# Creating indexes
add_index_to_students_email_column
```

2. Foreign Key Constraints:

```
Schema::create('enrollments', function (Blueprint $table) {
    $table->id();
```

3. Indexes for Performance:

```
Schema::create('students', function (Blueprint $table) {
    $table->id();
    $table->string('email')->unique();     // Unique index
    $table->string('name')->index();       // Regular index
    $table->integer('year');

    // Composite index
    $table->index(['major', 'year']);

    // Full-text search index
    $table->fullText(['name', 'bio']);
});
```

4. Safe Rollbacks:

```
public function up()
{
    Schema::create('students', function (Blueprint $table) {
        // Create table
```

Testing Relationships

Using Tinker:

```
php artisan tinker
```

Test Your Relationships:

```
// Create test data
>>> $student = Student::create(['name' => 'John', 'email' => 'john@test.com', 'major' => 'CS', 'year' => 2])
>>> $course = Course::create(['code' => 'CS101', 'name' => 'Intro to Programming', 'credits' => 3])

// Test many-to-many
>>> $student->courses()->attach($course->id, ['grade' => 3.5])
>>> $student->courses
>>> $course->students

// Test queries
>>> Student::with('courses')->get()
>>> Course::withCount('students')->get()
```

Database Seeding

Create Seeders:

```
php artisan make:seeder StudentSeeder
php artisan make:seeder CourseSeeder
```

StudentSeeder:

```php
<?php

use App\Models\Student;
use Illuminate\Database\Seeder;

class StudentSeeder extends Seeder
{
    public function run()
    {
        Student::create([
            'name' => 'John Doe',
            'email' => 'john@example.com',
            'major' => 'Computer Science',
            'year' => 2
        ]);
```

Run Seeders:

```
php artisan db:seed                          # Run all seeders
php artisan db:seed --class=StudentSeeder  # Run specific seeder
php artisan migrate:fresh --seed       # Fresh database with seed data
```

Register in DatabaseSeeder:

```php
<?php

use Illuminate\Database\Seeder;

class DatabaseSeeder extends Seeder
{
    public function run()
    {
        $this->call([
            StudentSeeder::class,
            CourseSeeder::class,
        ]);
    }
```

Performance Optimization

1. Use Indexes:

```
// Add indexes for frequently queried columns
$table->index('email');
$table->index(['major', 'year']);
```

2. Eager Loading:

```
// Bad: N+1 queries
$students = Student::all();
foreach ($students as $student) {
    echo $student->courses->count();
}

// Good: 2 queries
$students = Student::withCount('courses')->get();
foreach ($students as $student) {
    echo $student->courses_count;
```